codecademy

Learn R: Joining Tables

dplyr inner_join()

R data frame objects can be joined together with the dplyr function inner_join() . Corresponding rows with a matching column value in each data frame are combined into one row of a new data frame, and non-matching rows are dropped.

dplyr 's inner_join() takes two data frames as arguments and returns a new data frame with the corresponding rows merged together. Non-matching rows from each data frame are dropped in the resulting data frame.

For example, consider the sales and targets data frames of a t-shirt company. sales contains the monthly revenue for the company and has two columns: month and revenue . targets contains the goals for monthly revenue for each month and has two columns: month and target . To perform an inner join on the two data frames using dplyr :

```
sales_vs_targets <- sales %>%
  inner_join(targets)
```

inner_join() will use the month column as the column to match on, as both the sales and target data frames have a month column. The resultant data frame will only contain the matching rows from sales and targets .

Multiple data frames can be merged together at once by stringing multiple calls to inner_join with the pipe %>% .

For example, consider the same sales and targets data frames of a t-shirt company. An additional data frame small_medium_large contains the number of small, medium and large t-shirts sold per month and has four columns: month , small , medium , and large . To perform an inner join on the three data frames using dplyr :

```
sales_vs_targets <- sales %>%
  inner_join(targets) %>%
  inner_join(small_medium_large)
```

inner_join() will use the month column as the column to match on, as the sales , target , and small_medium_large data frames have a month column. The resultant data frame will only contain the

matching rows from `sales`, `targets`, and `small_medium_large`.

dplyr full_join()

In a full join, R data frame objects are merged together with the `dplyr` function `full_join()`. Corresponding rows with a matching column value in each data frame are combined into one row of a new data frame, and non-matching rows are also added to the resultant data frame with `NA`s for the missing information.

`dplyr`'s `full_join()` function will perform a full join, where non-matching rows are also added to the resultant merged data frame with `NA`s for the missing information.

For example, consider the inventory data frames of two stores, `store_a_inventory` and `store_b_inventory`. The `store_a_inventory` data frame contains two columns: `item` and `store_a_inventory`. The `store_b_inventory` data frame contains two columns: `item` and `store_b_inventory`. To perform a full join on the two data frames:

```
store_a_b_inventory <- store_a_in
  full_join(store_b_inventory)
```

The resultant data frame will contain each matching row from `store_a_inventory` and `store_b_inventory` as well as the non-matching rows from `store_a_inventory` and `store_b_inventory`.

dplyr bind_rows()

Multiple R data frames containing the same columns can be concatenated into one data frame using the dplyr function bind_rows() .

dplyr 's bind_rows() function takes all the data frames to bind as arguments and returns a single data frame where the data frames have been concatenated into a longer data frame.

For example, consider two customer data frames customer_1 and customer_2 , each containing columns name and email . To concatenate the data frames into one longer data frame:

```
customers <- customer_1 %>%
  bind_rows(customer_2)
```

If a third data frame customer_3 with columns name and email also existed, all three data frames could be concatenated into one longer data frame as follows:

```
customers <- customer_1 %>%
  bind_rows(customer_2) %>%
  bind_rows(customer_3)
```

dplyr join functions

R data frames can be joined on specific columns using one of the `dplyr` join functions and the `by` argument.

The `dplyr` join functions can take the additional `by` argument, which indicates the columns in the "left" and "right" data frames of a join to match on.

For example, consider the `orders` and `products` data frames of a business. The `orders` data frame contains five columns: `id`, `product_id`, `customer_id`, `quantity` and `timestamp`. The `products` data frame contains three columns: `id`, `product_id`, and `price`. To perform an inner join on the two data frames using `product_id` from the `orders` data frame and `id` from the `products` data frame as the columns to join on:

```
orders_products <- orders %>%
  inner_join(products,
                    by = c("produ
                    suffix = c("_
```

The `suffix` argument will append suffixes to column names that duplicate between the two data frames. `id` in the original `orders` data frame will become `id_orders` in the resultant data frame and `id` in the original `products` data frame will become `id_products` in the resultant data frame.

Efficient Data Storage with Multiple Tables

For efficient data storage, related information is often spread across multiple tables of a database. Consider an e-commerce business that tracks the products that have been ordered from its website. Business data for the company could be split into three tables:

- `orders` would contain the information necessary to describe an order: `order_id`, `customer_id`, `product_id`, `quantity`, and `timestamp`
- `products` would contain the information to describe each product: `product_id`, `product_description` and `product_price`
- `customers` would contain the information for each customer: `customer_id`, `customer_name`, `customer_address`, and `customer_phone_number`

This table structure prevents the storage of redundant information, given that each customer's and product's information is only stored once, rather than each time a customer places an order for another item.

Print Share